The Persistence Calculus: unique stable models for persistence rules

Hayashi, Hisashi

ISSN 1369-1961

Department of
Computer Science

Technical Report No. 750

The Persistemce Calculus: unique stable models for persistence rules

Hisashi Hayashi

QUEEN MARY
AND WESTFIELD COLLEGE
UNIVERSITY OF LONDON

September 1998

The Persistence Calculus: unique stable models for persistence rules

Hisashi Hayashi

Department of Computer Science,
Queen Mary and Westfield College,
University of London,
Mile End Road, London E1 4NS, U.K.
email: hisashi@dcs.qmw.ac.uk

Abstract

One of the problems with logic programs for temporal reasoning is that their semantics are not always unique. On the other hand, this paper presents a method for writing logic programs for temporal reasoning which guarantees a unique stable model. The ramification problem is also discussed from the view point of the number of the stable models. Comparison with the (simplified) event calculus is also done briefly.

Keywords: Temporal Reasoning, Logic Programming.

1 Introduction

Logic programs do not have unique semantics in general. Temporal reasoning in logic programming suffers the same problem. One of the treatments for this problem is to restrict programs so that they are *stratified* [1]. The stratification of a logic program is important in the sense that any stratified logic program has a unique stable model [3]. In this paper, a method to express temporal persistence rules in a stratified logic program is shown. The ramification problem is also discussed from the view point of the number of stable models. It is shown that by restricting the expression of indirect effects, it is still possible to stratify any program. The formalism of temporal reasoning in this paper is named *the persistence calculus.*

The idea of actions is not used any more. Instead, two kinds of fluents are introduced. One is a *continuous fluent* which continues to hold until it is stopped explicitly. The other one is a *non-continuous fluent* which holds at a time point

but does not continue afterwards. Actions which happen instantaneously can be regarded as non-continuous fluents. Therefore, compared with the logic programming version of the (simplified) event calculus [9], the persistence calculus can be regarded as a lower level language. It is possible to express in the persistence calculus that a fluent `f` holds immediately afterwards if two fluents `a1` and `a2` hold at the same time. If `a1` and `a2` are regarded as actions, this sentence expresses the effect of concurrent actions. Also, continuous changes can be expressed in the persistence calculus. Most of the techniques were used to translate a program in the higher level action language $\mathcal{H}$ [4, 5] into a stratified logic program.

The rest of the paper is organized as follows. After recapping the definition of stratification of logic programs in the next section, the persistence rule axioms and the domain description clauses are defined from Section 3 to Section 5. In Section 6, which is the most important section, it is shown that the set of clauses defined from Section 3 to Section 5 is stratified under some restrictions. In Section 7, some examples are shown. The conclusion is in Section 8.

2 Background

In this section, the stratification of logic programs is defined.

Definition 2.1 *A program P is* **stratified**[1] *if there exists a partition:*

$$P = P_1 \dot{\cup} ... \dot{\cup} P_n$$

such that the following conditions hold for any $i (1 \leq i \leq n)$:

- *if an atom* `a` *which is not preceded by "*`not`*" occurs in the body of a clause in P_i which defines* `b`*, then its definition is either in $\bigcup_{k=1}^{i} P_k$ or undefined*[2]*, which is denoted by* $\mathtt{b} \succeq \mathtt{a}$.
- *if an atom* `a` *which is preceded by "*`not`*" occurs in the body of a clause in P_i which defines* `b`*, then its definition is either in $\bigcup_{k=1}^{i-1} P_k$ or undefined, which is denoted by* $\mathtt{b} \succ \mathtt{a}$.

In this paper, $\mathtt{b} \equiv \mathtt{a}$ is defined as $(\mathtt{b} \succeq \mathtt{a}) \wedge \neg(\mathtt{b} \succ \mathtt{a})$.
The following theorem follows from [1] and [3].

Theorem 2.1 *If a program is stratified, there exists exactly one stable model*[3] *for the program.*

[1] Strictly speaking this definition of the stratification is extended from its original definition in [1] and called **local stratification**.

[2] Predicates whose unique truth values are evaluated by their external solvers are also defined as *undefined*.

[3] This stable model is consistent with the completion of the program.

3 Persistence rules

In the following persistence rules, two kinds of **fluents** are defined: 1. **non-continuous fluents**; 2. **continuous fluents**. A non-continuous fluent is a fluent which holds at a time point but does not hold afterwards. A continuous fluent is a fluent which continues to hold after a time point until it is stopped explicitly. Any fluent is either of the form "`pos(`x`)`" or of the form "`neg(`x`)`", where x is a term of logic programming. Intuitively, `neg(`x`)` means that x does not hold. Therefore, neg is an explicit negation. Note that a fluent can be both a continuous fluent and a non-continuous fluent. For example, a fluent which holds at the time point 5 and at all the time points between 10 and 1000 can be expressed easily.

The atom "`given0(`f,t`)`" is defined to express that the non-continuous fluent f holds at the time point t. The atom "`given+(`f,t`)`" means that the continuous fluent f continues to hold after the time point t until it is explicitly stopped continuing to hold. The atom "`released0(`f,t`)`" means that the continuous fluent f does not continue to hold at and after the time point t. The atom "`released+(`f,t`)`" means that the continuous fluent f does not continue to hold after the time point t. If both "`released0(`f,t`)`" and "`given0(`f,t`)`" are true, f is defined to hold at the time point t. Similarly, if both "`released+(`f,t`)`" and "`given+(`f,t`)`" are true, f is defined to continue to hold after the time point t.

These ideas are expressed by the following **persistence rule axioms**. These axioms extend the stratified persistence rule introduced by Evans and Sergot [2], where only one continuous fluent is considered for simplicity. Any query is supposed to be of the form "`?hold(`f,t`)`", which succeeds if the fluent f holds at the time point t.

```
hold(F,T):-given0(F,T).  [pa1]
hold(F,T2):-given+(F,T1), T1<T2, not broken(F,T1,T2).  [pa2]
broken(F,T1,T3):-released0(F,T2), T1<T2, T2<=T3.  [pa3]
broken(F,T1,T3):-released+(F,T2), T1<T2, T2<T3.  [pa4]
```

It is true that these persistence rules are only for *prediction*, but *explanation* can be calculated by abduction [10].

In order to stop automatically the continuous fluent `pos(x)` (`neg(x)`) continuing to hold when `neg(x)` (respectively `pos(x)`) holds, the following axioms are needed.

```
released0(pos(X),T):-given0(neg(X),T).  [pa5]
released0(neg(X),T):-given0(pos(X),T).  [pa6]
released+(pos(X),T):-given+(neg(X),T).  [pa7]
released+(neg(X),T):-given+(pos(X),T).  [pa8]
```

The problem is that if both "given0(pos(x),5)" and "given0(neg(x),5)" are true, both "hold(pos(x),5)" and "hold(neg(x),5)" are true. The same problem arises if both "given+(pos(x),5)" and "given+(neg(x),5)" are true. The latter case is pointed out in [2]. In this paper, nothing is done for this problem. Therefore, both pos(x) and neg(x) are assumed to hold in this case, avoiding making all the fluents true.

It is important to note that the above axioms are stratified as will be confirmed in Corollary 6.1.

4 Stratified domain description

Three kinds of **domain description clauses**, which are called **stratified domain description clauses**, are defined in this section. One clause expresses that if some predicates of logic programming are true, some fluents hold at a time point, and some fluents cannot be proved to hold at the time point, then another fluent starts to hold after the time point. This can be expressed by the clause of the following form, where F, F1,..., Fj, G1,..., Gk are fluents and P1,..., Pi are predicates of logic programming which are either true or false by the external solver.

```
given+(F,T):-P1, ..., Pi, hold(F1,T), ..., hold(Fj,T), not
   hold(G1,T), ..., not hold(Gk,T). [s1]
```

Note that "hold(neg(f),5)" and "not hold(pos(f),5)" are different. The former means that f does not hold at the time point 5. The latter means that f cannot be proved to hold. Therefore, it is possible to express that it is impossible to prove that f does not hold at the time point 5 by "not hold(neg(f),5)".

Another clause expresses that if some predicates of logic programming are true, some fluents hold at a time point, and some fluents cannot be proved to hold at the time point, then another fluent is stopped continuing to hold after the time point.

This can be expressed by the clause of the following form, where F, F1,..., Fj, G1,... Gk are fluents and P1,..., Pi are predicates of logic programming which are either true or false by the external solver.

```
released+(F,T):-P1, ..., Pi, hold(F1,T), ..., hold(Fj,T), not
   hold(G1,T), ..., not hold(Gk,T). [s2]
```

The other clause is for representing continuous changes based on Shanahan's approach [11].

The following clause expresses that if the fluent f starts to hold at the time point T1 and f is not stopped continuing to hold up to the time point T2, another

fluent pos(g(V))[4] holds at the time point T2(>T1), where V is a variable whose value *func*(T1,T2) is decided by the value of the two variables T1 and T2.

```
given0(pos(g(V)),T2):-given+(f,T1), T1<T2, not broken(f,T1,T2),
   V is func(T1,T2).   [s3]
```

For the readers who are not so familiar with Shanahan's treatment for continuous changes, an example is shown below.

Example 4.1 *The following program means that the height of water increases by 2cm a minute after the tap is opened.*

```
given0(pos(height(H)),T2):-given+(pos(open),T1), T1<T2, not
   broken(pos(open),T1,T2), H is 2 * (T2 - T1).
```

and [pa1]...[pa8]

As will be written in Corollary 6.1, these stratified domain description clauses of the form [s1], [s2], or [s3] together with the persistence rule axioms [pa1]...[pa8] are stratified.

5 Not always stratified clauses

Two kinds of other domain description clauses are defined in this section. Unlike stratified domain description clauses, the domain description clauses introduced in this section, together with the persistence rule axioms [pa1]...[pa8], are not always stratified, which means that the stable model is not always unique if these clauses are used. However, by restricting the use of these clauses, any program can be stratified.

The new type of domain description clauses are of the following form, where F, F1,..., Fj, G1, ..., Gk are fluents and P1, ..., Pi are predicates of logic programming which are either true or false by the external solver.

```
given0(F,T):-P1, ..., Pi, hold(F1,T), ..., hold(Fj,T), not
   hold(G1,T), ..., not hold(Gk,T). [ns1]

released0(F,T):-P1, ..., Pi, hold(F1,T), ..., hold(Fj,T), not
   hold(G1,T), ..., not hold(Gk,T). [ns2]
```

Compare these clauses with the stratified domain description clauses of the form [s1] and [s2]. The main difference is that while the heads of the clauses [ns1] and [ns2] mention the truth value of the fluent F at the time point T and the bodies

[4]For simplicity, the argument of g is assumed to be one. Technically speaking, however, it can have more than one argument.

of these clauses mention the truth value of the fluents F1,...,Fj, G1,...,Gk at the same time points T, the heads of the clauses [s1] and [s2] mention the truth value of F at the time points after T. Therefore, it is not difficult to imagine that the clauses of the form [ns1] and [ns2] cause the *ramification problem* and there might exist more than one stable model. In the next section, the method to restrict these domain description clauses will be introduced in order to have a unique stable model.

6 Stratification of the persistence calculus

In this section, a theorem is introduced which says that any program is stratified under some restrictions.

Before a method to stratify a program is introduced, the stratification of fluents is defined. Similar ideas [8] are used as a treatment for the ramification problem. Note that the stratification of fluents is different from the stratification of logic programs.

Definition 6.1 *The fluents in a program are* **stratified** *iff:*

- *any fluent belongs to exactly one stratum of fluents, where all the strata are totally ordered;*
- *for any x, the two fluents $pos(x)$ and $neg(x)$ belong to the same stratum.*
- *for any clause in the program of the form [ns1] or [ns2], the fluent* F *belongs to a higher stratum than the strata to which the fluents* F1,..., Fj, G1,..., Gk *belong.*

In this paper, f $\ll$ g denotes that the fluent g is defined in higher stratum than the stratum the fluent f belongs to. Now the theorem is introduced.

Theorem 6.1 *Let P be a program which is a set of the persistence rule axioms [pa1]...[pa8] and clauses of the form [s1], [s2], [s3], [ns1], or [ns2]. If the fluents in P are stratified, then P is stratified.*

Proof: A method to stratify the program is shown below.

1. First, the program can be split by time as follows. For all the time points T_{01}, T_{02}, $\mathbf{T_1}$ and $\mathbf{T_2}$ such that $\mathbf{T_1} < \mathbf{T_2}$ and for all the fluents F_1 and F_2, split the program so that all the atoms of the form:

   $\texttt{released+}(F_2,\mathbf{T_2})$, $\texttt{given+}(F_2,\mathbf{T_2})$, $\texttt{hold}(F_2,\mathbf{T_2})$, $\texttt{broken}(F_2,T_{02},\mathbf{T_2})$, $\texttt{released0}(F_2,\mathbf{T_2})$, $\texttt{given0}(F_2,\mathbf{T_2})$

are defined in higher strata than the strata to which the atoms of the form:

$$\texttt{released+}(\mathtt{F_1},\mathbf{T_1}),\ \texttt{given+}(\mathtt{F_1},\mathbf{T_1}),\ \texttt{hold}(\mathtt{F_1},\mathbf{T_1}),$$
$$\texttt{broken}(\mathtt{F_1},\mathtt{T_{01}},\mathbf{T_1}),\texttt{released0}(\mathtt{F_1},\mathbf{T_1}),\texttt{given0}(\mathtt{F_1},\mathbf{T_1})$$

belong.

2. Second, the program can be split by the fluents as follows. For all the time points $\mathtt{T_0}$ and $\mathtt{T_1}$ and for all the fluents $\mathbf{F_1}$ and $\mathbf{F_2}$ such that $\mathbf{F_1} \ll \mathbf{F_2}$, split the program so that all the atoms of the form:

$$\texttt{released+}(\mathbf{F_2},\mathtt{T_1}),\ \texttt{given+}(\mathbf{F_2},\mathtt{T_1}),\ \texttt{hold}(\mathbf{F_2},\mathtt{T_1}),$$
$$\texttt{broken}(\mathbf{F_2},\mathtt{T_0},\mathtt{T_1}),\texttt{released0}(\mathbf{F_2},\mathtt{T_1}),\texttt{given0}(\mathbf{F_2},\mathtt{T_1})$$

are defined in higher strata than the strata to which the atoms of the form:

$$\texttt{released+}(\mathbf{F_1},\mathtt{T_1}),\ \texttt{given+}(\mathbf{F_1},\mathtt{T_1}),\ \texttt{hold}(\mathbf{F_1},\mathtt{T_1}),$$
$$\texttt{broken}(\mathbf{F_1},\mathtt{T_0},\mathtt{T_1}),\texttt{released0}(\mathbf{F_1},\mathtt{T_1}),\texttt{given0}(\mathbf{F_1},\mathtt{T_1})$$

belong.

3. Third, the program can be split by predicates as follows. For all the time points $\mathtt{T_0}$ and $\mathtt{T_1}$ and for any fluent F, split the program so that:

$$\texttt{released+}(\mathtt{F},\mathtt{T_1}) \succ \texttt{given+}(\mathtt{F},\mathtt{T_1}) \succ \texttt{hold}(\mathtt{F},\mathtt{T_1}) \succ$$
$$\texttt{broken}(\mathtt{F},\mathtt{T_0},\mathtt{T_1}) \succ \texttt{released0}(\mathtt{F},\mathtt{T_1}) \succ \texttt{given0}(\mathtt{F},\mathtt{T_1})$$

Now it is easy to confirm that any program consisting of [pa1]...[pa8] and clauses of the form [s1][s2][s3] and [ns1][ns2] is stratified.

This theorem means that the stratification of the clauses is related to the stratification of the fluents mentioned in these clauses. Within this limitation, any program is stratified and does not suffer the ramification problem. the following corollary is straightforward to prove.

Corollary 6.1 *If P is a program which is a set of the persistence rule axioms [pa1]...[pa8] and clauses of the form [s1], [s2], or [s3], then P is stratified.*

7 Examples

Proposition 7.1 *The following program has a unique stable model.*

```
given+(pos(closed(a)),0).
given+(neg(closed(b)),0).
```

```
given+(neg(stuffy),0).
given0(pos(shut(b)),5).
given+(pos(closed(X),T):-hold(shut(X),T).
given0(pos(stuffy),T):-hold(pos(closed(a)),T),
   hold(pos(closed(b)),T).
given0(neg(stuffy),T):-hold(neg(closed(a)),T).
given0(neg(stuffy),T):-hold(neg(closed(b)),T).
```

and [pa1]...[pa8]

Proof: Because the fluents are stratified, the program is stratified by Theorem 6.1. By Theorem 2.1, the program has a unique stable model.

Proposition 7.2 *The following program is not stratified.*

```
given+(pos(stuffy),0).   [s1.1]
given0(neg(closed(a)),T):-hold(neg(stuffy),T),
   hold(pos(closed(b)),T).  [ns1.1]
given0(neg(closed(b)),T):-hold(neg(stuffy),T),
   hold(pos(closed(a)),T).  [ns1.2]
```

and [pa1]...[pa8]

Proof: In order to stratify [ns1.1] and [ns1.2],

(Lemma 1:) `given0(neg(closed(a)),T)` $\succeq$ `hold(pos(closed(b)),T)`
(Lemma 2:) `given0(neg(closed(b)),T)` $\succeq$ `hold(pos(closed(a)),T)`

must hold. In order to stratify [pa1]...[pa8],

(Lemma 3:) `hold(pos(closed(a)),T)` $\succ$ `given0(neg(closed(a)),T)`
(Lemma 4:) `hold(pos(closed(b)),T)` $\succ$ `given0(neg(closed(b)),T)`

must hold. By Lemma 1 and Lemma 3,

(Lemma 5:) `hold(pos(closed(a)),T)` $\succ$ `hold(pos(closed(b)),T)`

must hold. By Lemma 2 and Lemma 4,

(Lemma 6:) `hold(pos(closed(b)),T)` $\succ$ `hold(neg(closed(a)),T)`

must hold. Lemma 5 and Lemma 6 contradict. Therefore, this program is not stratified.

Note that the above fluents are not stratified.

8 Comparison with the event calculus

It is easy to understand that what can be expressed by the simplified event calculus [9] can be wriiten in the persistence calculus. Unless indirect effects of actions are expressed, all fluents in the simplified event calculus can be regarded as continuous fluents in the persistence calculus. Actions in the event calculus can be regarded as non-continuous fluents in the persistence calculus. Effects of (concurrent) actions and continuous changes in the event calculus can be expressed using clauses of the form [s1] and [s3] respectively in the persistence calculus. Indirect effects of actions (e.g. $holds(f,T) \leftarrow hold(g,T).$) in the event calculus can be expressed using clauses of the form [ns1].

One of the big difference between the persistence calculus and the event calculus is that while in the event calculus, fluents[5] which are defined by other fluents are not subject to the inertia, in the persistence calculus, a fluent can be both continuous and non-continuous. For example, in Proposition 7.1, `neg(stuffy)` is both continuous and non-continuous.

Another advantage to use the persistence calculus is the fact that it is easy to assimilate knowledge by observation. For example, if the robot observed that the car passed through the tunnel at time point 1000, then *given0(pos(at(car,tunnel)), 1000)* can be added to the knowledge base. If the robot observes that the block a is on the table at the time point 500, then *given+(pos(on(a,table)),1000)* can be added.

It is straightfoward that comparison with the situation calculus [7] can be done by allocating a time point to each situation. Comparison of the situation calculus and the event calculus is done by Kowalski and Sadri [6].

9 Conclusion

By Corollary 6.1, it was shown that any program consisting of the persistence rule axioms [pa1]...[pa8] and clauses of the form [s1]...[s3] is stratified and has a unique stable model. By Theorem 6.1, it was shown that any program consisting of [pa1]...[pa8] and clauses of the form [s1][s2][s3][ns1][ns2] is stratified if the fluents in the program are stratified. Therefore, the stratification of logic programs in the persistence calculus formalism is closely related to the stratification of fluents in the programs. Also, it was shown that the persistence calculus can express more than the event calculus can do as a lower level formalism.

[5]These fluents are called *derived* fluents by Lifshitz.

Acknowledgements

A lot of techniques in the present paper are based on the MSc thesis [4] of the author whose supervisor was Mr. Marek Sergot. The discussion with Dr. Murray Shanahan was also helpful.

References

[1] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, pages 89–148. 1988.

[2] D. Evans and M. Sergot. An abstract framework for the representation of persistence. Department of Computing, Imperial College, University of London, 1995.

[3] M. Gelfond and V. Lifschitz. The stable model for logic programming. In *JICSLP 86*, pages 1070–1080, 1986.

[4] H. Hayashi. The language H: A language for representing actions. Master's thesis, Department of Computing, Imperial College, University of London, 1996.

[5] H. Hayashi. Language $\mathcal{H}_{Simple}(\mathcal{R})$: an action language for representing concurrent actions and continuous changes. In *the second workshop on practical reasoning and rationality*, pages 1–13, 1997.

[6] R. Kowalski and F. Sadri. Reconciling the event calculus with the situation calculus. *Journal of Logic Programming*, 31:39–58, 1997.

[7] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, volume 4, pages 469–502. 1969.

[8] J. A. Pinto. Temporal reasoning in the situation calculus. Technical Report KRR-TR-94-1, Computer Science Department, University of Toronto, 1994.

[9] F. Sadri and R. Kowalski. Variants of the event calculus. In *ICLP*, pages 67–82. MIT Press, 1995.

[10] M. Shanahan. Prediction is deduction but explanation is abduction. In *IJCAI 89*, pages 1055–1060, 1989.

[11] M. Shanahan. Representing continuous change in the event calculus. In *ECAI 90*, pages 598–603, 1990.